

An Overview

Besides discussing various algorithms in CS3220 Data Structures and Algorithm Analysis, we also discussed how difficult it is to solve a problem. For example, to sort a list of n pieces of data, we have to do $\Omega(n)$ comparisons.

We now turn to another, more challenging, question: given a problem, whether or not a computer can solve it at all. More generally, *What can computers do and can't do?, and why?*

We will try to answer this question in terms of three related concepts: *computability, automata, and formal languages.*

Computability

In 1931, Kurt Gödel demonstrated that *within any given branch of mathematics, there would always be some propositions that couldn't be proved either true or false using the rules and axioms ... of that mathematical branch itself.*

This result is then used to conclude that: some problems are *unsolvable* by any “computers”, since any computer only knows a finite number of rules and axioms because of its finite nature.

An often used example of this nature is *Halting Problem*, i.e., there does not exist a program, H , that, given any program, P , and input, i , H can decide if P will not get into an infinite loop with i .

A tough job

It is already difficult enough to show that something is solvable: we have to find a solution. If it were easy, there would not need any homework, or exam.

It is even much more difficult to prove that something is unsolvable or not computable. Could we conclude that something is unsolvable, if we could not come up with a solution in 2 hours, or 10 days?

We could not, otherwise no instructor will be able to fail you in a course.

Maybe we have to work harder, or we are simply not smart enough.

Similarly, if we send in a program to calculate a function, but the computer does not come back with an answer in 10 days. Could we conclude this function is not calculable? Should we wait for some more time, say 150 years, or is it really not *computable* at all?

Maybe we should come up with another program, and/or try it in another computer.

Thus, a better question is that what do we mean by saying that a problem is not “computable” by a “computer”?

If we can provide a precise definition and, then, *prove* a problem is NOT solvable by a specific computer, we can conclude that it is unsolvable by *that computer*.

Automata and their uses

We will start with various mathematic models of computers, referred to as *automata*, in terms of its capability, from simpler ones (regular machines) to more complicated ones, and culminating at a precise model of computation, or algorithm, the Turing Machine.

In fact, we will show that TM is more powerful than any real computer since it has infinite amount of memory (processor vs. memory).

Notice that in this course, we only care about the *capability* of a machine but not even a bit on its *efficiency*.

The big deal is then, if a problem is not computable in TM then we can conclude that problem is not computable, no matter what computer is used.

Formal languages

A human being is (much) more capable as compared with a monkey, partly because the language she speaks is much more sophisticated.

Similarly, a good way to characterize the capability of an automaton is to specify the (formal) language it understands. What we will do in this part is to study what is exactly the language that different classes of automata can *generate* and/or *accept*.

This might be the more practical part of the theoretical computer science, as it finds wide application in the computer engineering, such as *compiler construction, hardware design, and artificial intelligence*.

A summary

We will define various automata based on their memory structure, then discuss the equivalence between these automata and different kinds of languages. This is to characterize exactly what “computers” we are talking about.

We will then discuss exactly what a specific computer can do and can't do in terms of its languages.

Because of the precise nature of such subjects, we will start with a (p)review of some of the mathematical concepts.